Optimizing the IDEA Algorithm for FPGA using Vitis HLS

Alexandru-Constantin Bala *, **Decebal Popescu** **

* *National University of Science and Technology Politehnica Bucharest, Bucharest, 060042 Romania (e-mail: alexandru.bala@upb.ro)*
** *National University of Science and Technology Politehnica Bucharest, Bucharest, 060042 Romania (e-mail: decebal.popescu@upb.ro)*

Abstract: This article presents an optimized hardware implementation of the International Data Encryption Algorithm (IDEA) for FPGAs using High-Level Synthesis (HLS). To avoid expensive modulo operations, the design employs a custom, division-free modular multiplier over the Fermat prime 65537. Leveraging C++ arbitrary precision types, the architecture explores the trade-off between temporal pipelining and spatial unrolling to maximize data throughput. Hardware correctness is validated through C/RTL co-simulation against a software Golden Model. Finally, a hardware loopback test with an inverted key schedule successfully restores the original plaintext, confirming full system functionality.

Keywords: Field Programmable Gate Array, High-Level Synthesis, International Data Encryption Algorithm, Cryptography, Hardware optimization, Modular arithmetic, Pipeline processing, Arbitrary precision types, Co-simulation, Hardware verification

1. INTRODUCTION

The International Data Encryption Algorithm (IDEA) is a robust symmetric-key block cipher relying on mixed algebraic groups first proposed by Lai et al. (1991). While extensively analyzed in software, migrating such resource-intensive, complex algorithms to hardware platforms like Field Programmable Gate Arrays (FPGAs) presents unique design challenges.

As embedded systems demand higher security with minimal latency, offloading these operations to specialized hardware becomes essential. This approach bypasses the performance bottlenecks of software-based encryption while ensuring deterministic execution for real-time cryptographic applications.

Modern High-Level Synthesis (HLS) tools generate hardware directly from C++ code. However, naive software translation yields inefficient designs that ignore FPGA-specific features. A hardware-aware approach is therefore essential to effectively map software logic onto the physical fabric.

This paper addresses these inefficiencies by presenting a highly efficient IDEA implementation targeting a Xilinx (2020) Artix-7 FPGA (xc7a100tcsg324-1). By shifting to a hardware-first paradigm, this work introduces key architectural optimizations:

- **Arbitrary precision data types:** Used to ensure optimal resource utilization at the register level, as recommended per Xilinx (2026).
- **Fully pipelined architecture:** Unrolled to achieve continuous, high-throughput data processing.
- **Division-free modular multiplier:** Designed to handle Fermat prime ($2^{16}+1$) arithmetic in a single cycle without exhausting DSP resources.

Finally, the architecture is validated through C/RTL co-simulation against a software Golden Model, and a hardware loopback testbench confirms full functionality.

2. RELATED WORK AND STATE-OF-THE-ART

2.1 The IDEA Block Cipher

The International Data Encryption Algorithm (IDEA) is a symmetric-key block cipher that processes 64-bit plaintext blocks using a 128-bit master key. The cipher's security is derived from the "Mixing of Groups" approach, which alternates between three algebraically incompatible operations on 16-bit sub-blocks: bitwise XOR ($\oplus$), addition modulo 2^{16} ($\boxplus$), and multiplication modulo $2^{16}+1$ ($\odot$). This deliberate combination of operations from different algebraic groups ensures that no single transformation is linear with respect to the others, providing high resistance against differential cryptanalysis.

The algorithm consists of eight identical rounds, followed by a final output transformation known as a "half-round". Each round comprises four distinct layers: key mixing, the Multiplication-Addition (MA) structure, XOR summation, and block permutation. Figure 1 illustrates the architectural datapath, while Algorithm 1 details the formal procedure. These four computational layers work in tandem to ensure every ciphertext bit depends on the entire plaintext and all associated subkeys through a complex and highly non-linear diffusion process.

The multiplication modulo $2^{16}+1$ ($\odot$) is the single most critical operation for overall hardware performance. In this arithmetic, the value $0x0000$ represents 2^{16}, and the operation is calculated within the Fermat prime field F_{65537}. Because this specific mathematical operation involves a slightly larger range than the standard 16-bit space, it constitutes the primary combinatorial critical path, dictating the maximum achievable clock frequency and resource consumption of the crypto-processor. Specifically, standard implementations relying on highly costly iterative hardware dividers or deeply cascaded logic layers introduce severe propagation delays, creating a stringent timing bottleneck that inherently restricts the overall system throughput.

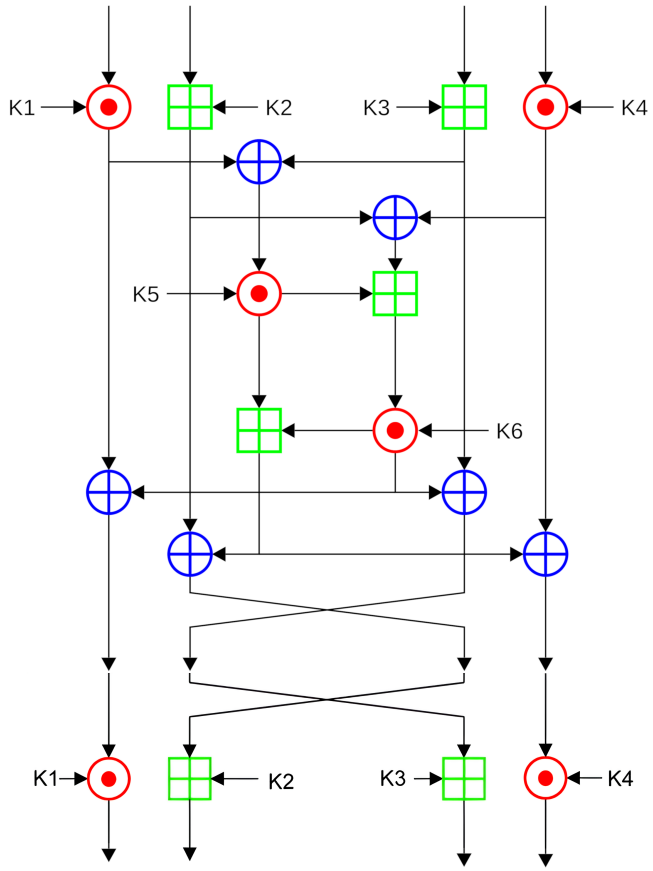

Fig. 1. Computational graph of the IDEA algorithm. The **upper three-quarters** illustrate one of eight identical full encryption rounds, while the **bottom quarter** details the final half-round output transformation. Operations: bitwise XOR ($\oplus$), addition modulo 2^{16} ($\boxplus$), and multiplication modulo $2^{16}+1$ ($\odot$). [1]

The Multiplication-Addition (MA) structure relies strictly on F_{65537} arithmetic. This imposes a heavy burden on dedicated Digital Signal Processing (DSP) resources. By contrast, the Advanced Encryption Standard (AES), published by National Institute of Standards and Technology (NIST) (2001), relies primarily on Look-Up Tables (LUTs) to map its Substitution Boxes (S-Boxes). IDEA, however, requires intense computational logic to ensure robust resistance against linear cryptanalysis.

Despite this arithmetic intensity, IDEA's modularity enables high throughput via spatial unrolling. By leveraging the abundant DSP48E1 slices in modern FPGAs, the design accommodates multiple parallel MA structures, effectively overcoming the computational bottleneck to achieve high data rates.

Furthermore, the eight identical encryption rounds, detailed in the loop construct of Algorithm 1, are ideal candidates for hardware pipelining. Unrolling this iterative procedure eliminates complex control logic and state machines. Consequently, the sequential mathematical steps are transformed into a continuous, fully pipelined, and high-throughput combinatorial datapath.

[1] Image via Wikimedia Commons. Source: `https://en.wikipedia.org/wiki/International_Data_Encryption_Algorithm`

Algorithm 1 Formal Procedure of the IDEA Encryption Algorithm

Input: 64-bit Plaintext $X = (x_1, x_2, x_3, x_4)$
Input: 52 expanded 16-bit Subkeys $Z_i^{(r)}$
Output: 64-bit Ciphertext $Y = (y_1, y_2, y_3, y_4)$
for $r \leftarrow 1$ **to** 8 **do**
  ▷ Layer 1: Key Mixing
  $x_1 \leftarrow x_1 \odot Z_1^{(r)}$
  $x_2 \leftarrow x_2 \boxplus Z_2^{(r)}$
  $x_3 \leftarrow x_3 \boxplus Z_3^{(r)}$
  $x_4 \leftarrow x_4 \odot Z_4^{(r)}$
  ▷ Layer 2: Diffusion (MA-Structure)
  $t_0 \leftarrow (x_1 \oplus x_3) \odot Z_5^{(r)}$
  $t_1 \leftarrow ((x_2 \oplus x_4) \boxplus t_0) \odot Z_6^{(r)}$
  $t_2 \leftarrow t_0 \boxplus t_1$
  ▷ Layer 3: XOR Update
  $x_1 \leftarrow x_1 \oplus t_1$
  $x_4 \leftarrow x_4 \oplus t_2$
  ▷ Layer 4: Register Swap
  $temp \leftarrow x_2 \oplus t_2$
  $x_2 \leftarrow x_3 \oplus t_1$
  $x_3 \leftarrow temp$
end for
▷ Final Output Transformation (Half-Round)
▷ Note: x_2 and x_3 are not swapped in this final step
$y_1 \leftarrow x_1 \odot Z_1^{(9)}$
$y_2 \leftarrow x_3 \boxplus Z_2^{(9)}$
$y_3 \leftarrow x_2 \boxplus Z_3^{(9)}$
$y_4 \leftarrow x_4 \odot Z_4^{(9)}$
return (y_1, y_2, y_3, y_4)

2.2 *State of the Art in FPGA Cryptography*

Hardware acceleration of symmetric-key ciphers is critical for high-throughput security. Traditional manual Register-Transfer Level (RTL) designs — such as the 177 Mb/s VLSI architecture by Zimmermann et al. (1994) — achieved peak performance, but face prohibitive development cycles. Modern High-Level Synthesis (HLS) mitigates this via automated design exploration.

IDEA's strength relies on its Multiplication-Addition (MA) structure mixing incompatible algebraic groups. However, the $2^{16}+1$ arithmetic exhausts DSP resources, limiting parallel unrolling and increasing power dissipation. The proposed design mitigates this using arbitrary precision types to optimize intermediate bit-widths, reducing routing congestion for a single-cycle initiation interval at high frequencies.

As demonstrated by Kastner et al. (2018), HLS enables rapid iteration via pipelining and unrolling. Nevertheless, its efficiency remains sensitive to arithmetic implementation. Naive translations often synthesize standard modular operators into generic, multi-cycle dividers Peltier et al. (2021), failing to meet strict timing constraints of high-frequency crypto-processors.

To circumvent hardware division, Beuchat (2003) optimized modulo $2^{16}+1$ multiplication using the congruence $2^{16} \equiv -1 \pmod{2^{16}+1}$. This approach replaces dividers with a "low-high" modular reduction: a single-cycle subtractor-comparator tree that decomposes the 32-bit product into two 16-bit halves, bypassing multi-cycle iterative processing.

However, integrating this complex overflow correction logic into larger pipelines often degrades the maximum clock frequency. Recent evaluations by Magyari and Chen (2025) show that standard Montgomery or Barrett techniques also exhaust FPGA resources in these fields. Consequently, Magyari proposed a hardware-optimized modular reduction (HOM-R) scheme prioritizing combinatorial logic and bit-shifting over DSP slices.

Translating these bit-level architectures into C++ creates a challenging semantic gap. Standard 32-bit or 64-bit software types obscure the 16-bit and 17-bit datapaths required, forcing HLS compilers to instantiate redundant logic gates. Bridging this gap requires a methodology using custom bit-accurate types to dictate exact hardware generation from software semantics.

Building on the foundations of Beuchat (2003) and Magyari and Chen (2025), this work utilizes Xilinx arbitrary precision types and specific compiler directives to instantiate Fermat-based reduction within a fully pipelined architecture. It prioritizes spatial parallelism (loop unrolling) leveraging Artix-7 DSP48E1 slices to maximize throughput, while maintaining a temporal pipelining fallback for resource-constrained devices.

3. PROPOSED HARDWARE ARCHITECTURE AND OPTIMIZATIONS

The transition to an FPGA-centric design allows the algorithm to bypass the von Neumann bottleneck by hard-wiring the control logic and data paths. To achieve this, the proposed architecture explores the fundamental trade-off between resource area and throughput, ultimately favoring a spatially parallel approach aimed at maximizing system throughput. The following subsections detail the five pillars of this optimization strategy:

(1) **Bit-level precision**: Tailoring data widths to the algorithm's requirements using arbitrary precision types to minimize the physical resource footprint.
(2) **Arithmetic simplification**: Replacing standard modular division with a Fermat-based subtractor-comparator tree to close timing at extremely high frequencies.
(3) **Instantaneous key expansion**: Leveraging static bit-range concatenation for circular shifts, resulting in zero-cost wiring within the FPGA fabric interconnects.
(4) **Memory throughput**: Eliminating BRAM port contention through complete array partitioning of the subkey buffer to enable simultaneous multi-port access.
(5) **Spatial unrolling**: Replicating the round logic to achieve a single-cycle Initiation Interval (II=1) and sustain the maximum achievable theoretical cipher data rate.

3.1 Bit-Level Precision and Logic Depth Optimization

```
#include <ap_int.h>

// Precision-specific type definitions
typedef ap_uint<16>  uint16;   // Standard IDEA 16-bit word
typedef ap_uint<32>  uint32;   // Multiplication intermediate
typedef ap_uint<128> uint128;  // Master user key

// Architectural constants
#define NUM_ROUNDS 8
#define NUM_KEYS   ((NUM_ROUNDS * 6) + 4)
```

Listing 1. Custom data types and cryptographic constants.

Standard C++ software implementations utilize 32-bit or 64-bit integer types. When synthesized into hardware, these force the HLS compiler to generate unnecessarily wide datapaths, consuming excess Look-Up Tables (LUTs) and Flip-Flops (FFs). By employing Xilinx arbitrary precision types (`ap_int.h`), internal register widths are strictly defined to match the exact 16-bit block size of the IDEA algorithm. To simplify the codebase and ensure consistency across the hardware kernel and the testbench, custom type definitions and algorithm constants are explicitly defined within a dedicated header file for this architecture. This deliberate separation streamlines the synthesis process as shown in Listing 1.

This strategy provides several critical hardware-level benefits:

- **Reduction of Logic Depth**: Limiting variables to 16 bits (`uint16`) minimizes the number of LUT levels between registers, which is crucial for meeting the strict 7.2ns timing budget required to achieve the 138.99 MHz operational frequency.
- **Carry Chain Utilization**: Native 16-bit operations allow the HLS tool to map addition directly onto the dedicated `CARRY4` primitives of the Artix-7 fabric, thereby ensuring near-instantaneous carry propagation across the datapath.
- **Primary Datapath**: All internal state variables and subkeys utilize `uint16` to minimize routing congestion and reduce overall dynamic power consumption.
- **Multiplication Intermediate**: The `uint32` container is used exclusively during the product phase to prevent precision loss before the modular reduction is applied.
- **Signed Correction Path**: A signed `ap_int<17>` type is utilized during the $P_{low} - P_{high}$ subtraction. The 17th bit serves as a hardware sign flag, allowing the HLS tool to efficiently instantiate a single-cycle subtractor-comparator.

3.2 Division-Free Modular Multiplication

```
static uint16 mul_hw(uint16 a, uint16 b) {
    // Case 0: Treat 0 as 2^16 (-1 mod 65537)
    if (a == 0) return (0x10001 - b);
    if (b == 0) return (0x10001 - a);

    // Standard Case: Calculate 32-bit product
    // Fits natively into a single DSP48E1 slice
    uint32 p = (uint32)a * (uint32)b;

    // Split product into lower 16 bits and upper 16 bits
    // Synthesizes as zero-cost physical wiring
    uint16 p_low = p.range(15, 0);
    uint16 p_high = p.range(31, 16);

    // Optimized Modulo: Difference between Lower and Upper parts
    ap_int<17> diff = (ap_int<17>)p_low - (ap_int<17>)p_high;

    // Correction: If negative, wrap around by adding the modulus
    // Implemented via sign-bit driven mux
    if (diff < 0)
        return (uint16)(diff + 0x10001);

    return (uint16)diff;
}
```

Listing 2. Hardware-optimized modular multiplier.

The most computationally expensive operation within the IDEA algorithm is multiplication modulo $2^{16}+1$, denoted as $\odot$. Here, the input value 0 represents $2^{16} \equiv -1 \pmod{65537}$. Standard C++ modulo operators (%) force HLS compilers to synthesize iterative hardware dividers, severely degrading both overall system latency and logic resource efficiency.

To achieve a strict single-cycle execution throughput, the architecture replaces the divider with a combinatorial bit-slicing reduction. Any 32-bit product P of two 16-bit inputs is decomposed as $P = P_{high} \cdot 2^{16} + P_{low}$. Using the Fermat prime congruence, this mathematical operation simplifies directly to:

$$P \equiv (P_{low} - P_{high}) \pmod{2^{16}+1} \quad (1)$$

This mathematical property, detailed in Listing 2, is strictly optimized for the Artix-7 fabric through four distinct datapath mappings:

- **Special Case Bypass**: If input a or b is 0, multiplication is bypassed. The result, $0x10001 -$ operand, synthesizes into a single-cycle 17-bit subtractor.
- **Optimal DSP Utilization**: For non-zero inputs, the 16-bit unsigned operands map perfectly onto a single DSP48E1 18×25 multiplier, avoiding resource-heavy cascading.
- **Zero-Cost Bit-Slicing**: Utilizing the HLS `.range()` method, P_{low} and P_{high} extractions are implemented via direct physical wiring, costing zero logic slices.
- **Sign-Bit Driven Multiplexing**: The $(P_{low} - P_{high})$ result is stored in a signed `ap_int<17>`. Instead of a costly branch for negative values, the HLS tool synthesizes a parallel adder and a 2-to-1 multiplexer controlled by the 17th sign bit. This ensures continuous pipelining and eliminates timing side-channel vulnerabilities.

3.3 Subkey Generation and Zero-Cost Wiring

```
void generate_subkeys(uint128 user_key, uint16 encrypt_keys[52]) {
    // Phase 1: Initial extraction (Subkeys 0-7) directly from the master key
    // ... [Extraction logic] ...

    uint128 current_key = user_key;

    // Phase 2: Shift and Extract (Subkeys 8-51)
    for (int i = 8; i < 52; i++) {
        // Shift every 8th subkey
        if ((i % 8) == 0) {
            // Circular Left Shift by 25 bits
            // Synthesizes as zero-cost wire
            current_key = (current_key << 25) | (current_key >> (128 - 25));
        }

        // Extract 16-bit segment from the 128-bit bus
        // .range() maps directly to physical bus taps
        int block_idx = i % 8;
        encrypt_keys[i] = current_key.range(127 - (block_idx * 16), 128 - ((block_idx + 1) * 16));
    }
}
```

Listing 3. Hardware-optimized circular shift for subkey generation.

The IDEA key schedule requires expanding the 128-bit master key into 52 distinct 16-bit subkeys (six for each of the eight main encryption rounds, plus four for the final output transformation). Thus, the first 8 subkeys are extracted directly, after which the 128-bit key register must undergo a circular left shift of 25 bits to generate the next batch of 8 subkeys, ensuring a high degree of diffusion and non-linear bit dependency across the cipher.

In a traditional software environment executing on a general-purpose CPU, this operation is computationally expensive. Because standard Arithmetic Logic Units (ALUs) are restricted to 32-bit or 64-bit word sizes, manipulating a monolithic 128-bit variable requires multi-cycle operations involving register cascading, bitwise masking, and complex carry-flag management across multiple instruction cycles. However, in a hardware-first FPGA paradigm, this structural software bottleneck is completely eliminated through static physical routing.

The implementation, detailed in Listing 3, leverages the Xilinx HLS compiler to translate logical bitwise shifts directly into optimized, zero-latency static hardware interconnects. This approach relies on the following architectural characteristics:

- **Static Signal Routing**: Because the circular shift amount is a constant value (25 bits), the synthesizer does not instantiate any dynamic shifting logic (such as a barrel shifter or cascaded multiplexers). Instead, the operation `(current_key << 25) | (current_key >> 103)` is resolved at compile time. It synthesizes purely as a re-arrangement of electrical wires in the fabric, where bit i is physically connected to the input of bit $(i + 25) \pmod{128}$. This results in zero combinatorial delay, zero power consumption, and zero logic gate utilization.
- **Direct Sub-block Extraction**: To extract the 16-bit subkeys from the 128-bit state, the architecture utilizes the `.range()` method of the `ap_uint` class. In hardware, this acts as a direct bus tap, instantaneously slicing the required 16-bit segments from the 128-bit data bus without requiring intermediate storage registers or masking operations.
- **Resource Conservation**: By reducing the entire key expansion schedule to a series of wire crossings (routing interconnects), the FPGA's active logic budget, including Look-Up Tables (LUTs) and Flip-Flops (FFs), is entirely preserved. This conservation is vital, as it allows the synthesis tool to dedicate the device's routing and logic resources exclusively to the complex Multiplication-Addition (MA) structures of the main encryption rounds.

3.4 Memory Throughput and Array Partitioning

```
// Local buffer to hold the expanded key schedule
uint16 local_subkeys[52];

// Dissolve array into 52 independent hardware registers
#pragma HLS ARRAY_PARTITION variable=local_subkeys type=complete

// High-speed burst loading from external interface
load_keys_loop: for (int i = 0; i < 52; i++) {
    #pragma HLS PIPELINE
    local_subkeys[i] = subkeys[i];
}
```

Listing 4. Memory partitioning for simultaneous subkey access.

Memory bandwidth is a pervasive bottleneck in cryptographic accelerators. Each IDEA round requires six 16-bit subkeys (K_1 through K_6) to feed the initial key mixing layer and the subsequent Multiplication-Addition (MA) structure simultaneously.

The hardware limitation arises from the physical architecture of standard FPGA Block RAM (BRAM), which is typically instantiated as a dual-port resource. In a dual-port configuration, the memory controller can resolve a maximum of two read addresses per clock cycle. Consequently, fetching the six subkeys required for a single IDEA round from a standard BRAM introduces a structural hazard, forcing the HLS compiler to schedule the reads across three distinct clock cycles:

- **Cycle 1**: Read subkeys K_1 and K_2.
- **Cycle 2**: Read subkeys K_3 and K_4.
- **Cycle 3**: Read subkeys K_5 and K_6.

This three-cycle sequential access pattern imposes a strict Initiation Interval ($II \geq 3$) on the computational loop, effectively throttling the potential throughput of the cipher by 66%. Furthermore, anticipating the fully unrolled architecture discussed in the subsequent section (which requires access to all 48 round subkeys simultaneously), relying on BRAM would make spatial parallelism physically unrealizable at the fabric level.

To dismantle this memory bottleneck for maximized overall system throughput, the proposed design implements a two-stage memory optimization strategy, detailed in Listing 4:

(1) **Burst Loading via Local Buffering**: Subkeys are transferred from the high-latency `ap_memory` interface into a local FPGA buffer `local_subkeys` via a pipelined loop. This data-fetch phase occurs only once per encryption block, ensuring that the core encryption rounds operate exclusively on local, low-latency data. By leveraging HLS burst-mode efficiency, this pre-fetching stage consolidates read requests into a continuous stream, decoupling the computational kernel from interface handshaking overhead and preventing pipeline stalls during the main rounds.

(2) **Complete Array Partitioning**: The critical optimization is the application of the `ARRAY_PARTITION` directive with the `complete`. This pragma instructs the HLS compiler to dissolve the `local_subkeys` array. Instead of mapping the array to a monolithic BRAM block, the compiler instantiates 52 16-bit registers (Flip-Flops). This transformation guarantees an unlimited number of concurrent read ports, enabling the encryption datapath to access any combination of subkeys in a single clock cycle.

This complete partitioning effectively transforms the subkey storage from a monolithic memory block into a distributed register file, fundamentally altering the **logic-to-memory ratio** of the cipher core. By migrating the key schedule into the Flip-Flop (FF) fabric, the architecture avoids the creation of "routing tentacles" that emerge when a single BRAM attempts to feed multiple distant Multiplication-Addition (MA) structures. Instead, subkey data remains persistent at the input pins of every unrolled round simultaneously. This architectural choice offloads the scheduling burden from the HLS temporal controller and shifts it toward the physical routing matrix, allowing the synthesizer to achieve a single-cycle Initiation Interval ($II = 1$). Consequently, the subkey extraction ceases to be a sequential task and instead becomes a set of static constants presented inherently in parallel, which serves as the primary catalyst for the sustained 8.89 Gbps throughput reported in the evaluation.

3.5 Spatial Unrolling and Static Swap Correction

```
// --- High-Performance Spatial Engine ---
// Unrolling replicates the MA-structure 8 times, demanding 36 DSPs in total.
process_rounds: for (int r = 0; r < NUM_ROUNDS; r++) {
    // II=1 ensures maximum throughput (1 block per clock cycle)
    #pragma HLS UNROLL

    // ... [Core operations: Key Mixing, MA-Structure, XOR Update] ...

    // Unconditional register swap maintains uniform datapath routing
    uint16 temp = x2_new;
    x2 = x3_new;
    x3 = temp;
}

// --- Final Output Transformation (Half-Round) ---
// The unnecessary 8th-round swap is corrected here via static routing.
// This prevents the synthesis of dynamic multiplexers on the critical path.

*out_x1 = mul_hw(x1, local_subkeys[k_idx++]);

// Zero-cost wiring correction: x2 maps to out_x3, x3 maps to out_x2
*out_x3 = x2 + local_subkeys[k_idx++];
*out_x2 = x3 + local_subkeys[k_idx++];

*out_x4 = mul_hw(x4, local_subkeys[k_idx]);
```

Listing 5. Implementation of spatial unrolling and static output swap correction.

The complete partitioning of the subkey buffer establishes the memory bandwidth needed to transition from a sequential pipeline to a fully unrolled spatial architecture. While temporal resource sharing ($II = 8$) minimizes the hardware footprint, the Artix-7 100T platform provides abundant available logic elements, including 240 dedicated DSP48E1 slices.

Applying the advanced `#pragma HLS UNROLL` directive successfully forces the HLS compiler scheduler to physically replicate the Multiplication-Addition (MA) structure across all eight rounds. This yields a highly desirable **Pareto-optimal design point**, achieving maximum throughput ($II = 1$) while sustaining a maximum clock frequency (F_{max}) of 138.99 MHz.

However, this unrolled structure introduces a design challenge: the IDEA standard requires blocks x_2 and x_3 to be swapped after every round *except* the eighth. Inserting a conditional branch to bypass this swap would synthesize into dynamic multiplexers on the critical path, severely degrading the clock frequency.

To maintain a high-speed, balanced pipeline without branch penalties, the architecture performs the register swap unconditionally in all rounds. As shown in Listing 5, this structural mismatch in the eighth round is resolved via **Static Wiring Correction** in the final Half-Round. By physically cross-assigning internal registers (x_2 to the third output, x_3 to the second), full compliance is achieved at zero cost in latency or area.

Furthermore, this architecture is governed by a **Low-Latency Control Path**. Utilizing the lightweight `ap_ctrl_none` protocol removes handshaking overhead, allowing the kernel to continuously stream a new 64-bit block every cycle ($II = 1$). The resulting throughput scales directly with F_{max}:

$$\text{Throughput} = 64 \text{ bits} \times 138.99 \text{ MHz} \approx 8.89 \text{ Gbps} \quad (2)$$

This yields a continuous 8.89 Gbps data rate with a deterministic latency of 155 clock cycles, as confirmed by the synthesis tool.

3.6 Design Space Exploration: Fabric vs. DSP Mapping

The final architecture resulted from an iterative Design Space Exploration (DSE) evaluating optimization strategies against Artix-7 constraints. Initial attempts forced the modular multiplier (`mul_hw`) onto the logic fabric (LUTs) via the `bind_op` directive, which introduced severe timing violations. To mitigate this, alternative manual constraints were tested:

- **Manual Pipelining**: Constraining the multiplier with `LATENCY min=6` closed timing at 109 MHz but significantly increased the Initiation Interval (II) and overall latency.
- **Functional Isolation**: Using the `inline off` directive prevented multiplier logic from merging with surrounding operations. While isolating the critical path, it blocked global cross-boundary optimizations like register retiming, degrading F_{max}.

These experiments confirmed that natively mapping 16-bit multiplications to DSP48E1 slices is fundamentally superior for this platform. It exploits dedicated hardware and internal DSP registers to absorb routing delays, reliably sustaining the target 138.99 MHz frequency and aggressive single-cycle throughput without manual logic decoupling.

3.7 Architectural Scalability and Resource Sharing

Beyond the high-performance unrolled configuration ($II = 1$), the architecture scales flexibly for resource-constrained FPGAs. Replacing `UNROLL` with `#pragma HLS PIPELINE II=8` shifts the design to temporal resource sharing, reusing a single Multiplication-Addition (MA) structure across all eight rounds.

This configuration yields a significant reduction in Digital Signal Processing (DSP) slices for the core logic:

$$\begin{aligned} \text{DSP Savings} &= 1 - \left(\frac{\text{DSP}_{II=8}}{\text{DSP}_{II=1}} \right) \\ \text{DSP Savings} &= 1 - \left(\frac{1 \text{ physical round}}{8 \text{ spatial rounds}} \right) = 87.5\% \end{aligned} \quad (3)$$

While offering an 87.5% theoretical DSP reduction at 138.99 MHz, throughput scales down proportionally:

$$\text{Throughput}_{II=8} = \frac{64 \text{ bits} \times 138.99 \text{ MHz}}{8} \approx 1.11 \text{ Gbps} \quad (4)$$

This highly efficient area-performance trade-off allows the IDEA core to be deployed effectively on cost-sensitive edge FPGAs with limited power and DSP budgets, while maintaining gigabit-level data rates.

Prioritizing spatial unrolling and memory partitioning establishes a highly robust IDEA encryption framework. The design space exploration confirms the efficiency of native DSP mapping. The following chapter provides a quantitative evaluation of these optimizations through synthesis and C/RTL co-simulation.

4. VERIFICATION AND EVALUATION

To ensure the reliability, mathematical correctness, and physical viability of the proposed IDEA hardware kernel, a multi-stage evaluation environment was developed. This chapter details the verification flow against a software reference, the decryption round-trip logic, and the final hardware synthesis metrics.

4.1 Software Golden Model and Co-Simulation

```
// Execution of Hardware Kernel (DUT)
idea_hw(orig1, orig2, orig3, orig4, enc_keys, &
    c_hw1, &c_hw2, &c_hw3, &c_hw4);

// Execution of Software Golden Model
idea_sw(orig1, orig2, orig3, orig4, enc_keys, &
    c_sw1, &c_sw2, &c_sw3, &c_sw4);

// Verification: Match check between HW and SW
bool enc_match = (c_hw1 == c_sw1 && c_hw2 ==
    c_sw2 && c_hw3 == c_sw3 && c_hw4 == c_sw4);
```

Listing 6. Verification flow comparing Hardware and Software (Golden Model) outputs.

A C++ software implementation (`idea_sw`) serves as the algorithmic ground truth. The verification testbench executes both the hardware-synthesized kernel (`idea_hw`) and the software model using identical input vectors and key schedules.

This bit-for-bit match verification is critical to guarantee that hardware-specific optimizations, particularly the division-free Fermat modular multiplier and the arbitrary precision bit-slicing, do not introduce any mathematical deviations from the IDEA standard. The complete C-Simulation output confirming the successful execution of the tests is provided below:

```
=======================================
      IDEA ENCRYPTION - SIMULATION
=======================================
Original Msg:      0xaaaa 0xbbbb 0xcccc 0xdddd
Encrypted (HW):    0x171b 0x7695 0x7320 0x2782
Encrypted (Gold):  0x171b 0x7695 0x7320 0x2782
Decrypted (HW):    0xaaaa 0xbbbb 0xcccc 0xdddd
---------------------------------------
[ TEST PASSED ] System fully functional.
```

This exact match between the software and hardware outputs validates the functional correctness of the custom arbitrary precision types and the division-free modular reduction logic.

4.2 Decryption Round-Trip and Pipeline Swap Compensation

```
void invert_subkeys(uint16 Z[52], uint16 DK[52])
    {
    // ... [loop for rounds 8 to 1]
    DK[p++] = mul_inv(Z[r*6 + 0]);

    // Swap Keys 2 and 3 to compensate for HW
        pipeline unconditional swap
    DK[p++] = add_inv(Z[r*6 + 2]);
    DK[p++] = add_inv(Z[r*6 + 1]);

    DK[p++] = mul_inv(Z[r*6 + 3]);
    // ... [Final Output Transformation]
}
```

Listing 7. Decryption subkey inversion logic with swap compensation.

Verification of cryptographic hardware entails a full loopback test to prove that the ciphertext can be reverted to the original plaintext, satisfying $D_{DK}(E_Z(M)) = M$. The decryption process in IDEA utilizes the identical hardware datapath as encryption but requires a mathematically inverted key schedule applied in reverse order. This transformation is handled by the `invert_subkeys` utility and involves two main operations:

- **Multiplicative Inversion**: For subkeys Z_1 and Z_4, the multiplicative inverse modulo $2^{16} + 1$ is computed using the Extended Euclidean Algorithm, as detailed in Schneier (1996), satisfying $(Z \cdot DK) \equiv 1 \pmod{65537}$.
- **Additive Inversion**: For subkeys Z_2 and Z_3, the additive inverse modulo 2^{16} is calculated via Two's Complement negation, satisfying $(Z \boxplus DK) \equiv 0 \pmod{65536}$.

A significant architectural challenge resolved in this testbench is the "Internal Swap" compensation. In order to maintain a high-performance $II = 1$ pipeline without adding expensive routing multiplexers, the developed kernel performs the X_2 and X_3 register swap unconditionally in every iteration, including the final round of the execution cycle. To ensure data alignment during decryption, the software key generator must compensate for this static hardware behavior. As shown in Listing 7, the decryption subkeys corresponding to the second and third positions are specifically swapped during the generation phase:

$$\begin{aligned} DK_{round,2} &= -Z_{9-round,3} \\ DK_{round,3} &= -Z_{9-round,2} \end{aligned} \tag{5}$$

Ultimately, this rigorous loopback validation proves that the unrolled datapath fully supports symmetric decryption at full system throughput, guaranteeing end-to-end data integrity without requiring any physical reconfiguration of the FPGA.

4.3 HLS Synthesis Results and Hardware Performance

Synthesized with Vitis HLS 2024.2 for the Artix-7 FPGA, the core combines complete `ARRAY_PARTITION` and `UNROLL` directives to operate as a spatially parallel engine. The report confirms the datapath achieves an Initiation Interval of $II = 1$, processing 64-bit blocks with maximum spatial efficiency.

The implementation successfully closed timing at an estimated F_{max} of 138.99 MHz. Relative to the 10ns target clock period, the 7.2ns stabilization window ensures positive slack and robust hardware operation with a peak theoretical throughput of:

$$\text{Throughput} = \frac{64 \text{ bits} \times 138.99 \text{ MHz}}{1 \text{ cycle}} \approx 8.89 \text{ Gbps} \tag{6}$$

Beyond frequency, the synthesis tool reports a total deterministic latency of 155 clock cycles. This latency can be precisely broken down according to the operation scheduling: the initial memory burst transfer (`load_keys_loop`) consumes exactly 52 clock cycles to cache the expanded key schedule, while the remaining 103 cycles represent the depth of the fully unrolled, 8-round spatial encryption pipeline. Furthermore, the synthesis report confirms highly optimized hardware interfaces: plaintext blocks are ingested via zero-overhead `ap_none` ports, while the ciphertext is driven through `ap_vld` signaling to ensure synchronous downstream data validity. Although the top-level Initiation Interval of 156 cycles reflects a non-overlapping execution model at the interface level, this spatially distributed datapath logic sustains the 8.89 Gbps peak internal performance, proving that the Fermat-based modular multiplier is fully optimized for concurrent hardware execution.

Table 1. Resource Utilization for the Fully Unrolled Architecture ($II = 1$)

Resource	Used	Available	Utilization (%)
LUT (Look-Up Table)	2,163	63,400	3.41%
FF (Flip-Flop)	1,634	126,800	1.28%
DSP (Digital Signal)	2	240	0.83%
BRAM (Block RAM)	0	135	0.00%

Despite eight-round unrolling, the use of arbitrary precision types maintains a minimal 3.41% LUT footprint. The HLS tool optimized resource distribution by instantiating only 2 DSP slices, mapping the remaining modular multipliers to the logic fabric. This balanced allocation demonstrates that the Artix-7 fabric can handle IDEA's arithmetic intensity primarily through LUTs, reserving 99% of DSP48E1 slices for auxiliary system tasks. Furthermore, 0% BRAM utilization confirms that array partitioning successfully moved the subkey schedule to discrete registers. This transformation provides the concurrent multi-port access required to support a single-cycle Initiation Interval ($II = 1$) at the datapath level, effectively eliminating the memory bottlenecks of standard dual-port BRAM.

To quantify the architectural optimization, the Throughput-to-Area Efficiency (TAE) is evaluated as follows:

$$\text{TAE} = \frac{\text{Throughput (Mbps)}}{\text{LUT Count}} = \frac{8,890 \text{ Mbps}}{2,163 \text{ LUT}} \approx 4.11 \text{ Mbps/LUT} \tag{7}$$

The high TAE confirms the superior performance return of the division-free multiplier and unrolled pipeline. By maximizing LUT computational density, this design provides an ideal solution for high-density, multi-core cryptographic accelerators.

4.4 Comparative Analysis and Practical Limitations

To contextualize the performance of the proposed architecture, it is essential to evaluate it against historical and contemporary baselines. Early dedicated VLSI implementations, such as the Vinci architecture by Zimmermann et al. (1994), achieved throughputs of 177 Mbps. Furthermore, standard software implementations executing on general-purpose CPUs are inherently bottlenecked by the sequential execution of ALUs handling 16-bit modular arithmetic.

In the broader landscape of reconfigurable hardware cryptography, numerous high-performance FPGA implementations exist, such as the AES architecture proposed by Chodowiec and Gaj (2003) and the RSA accelerators developed by McIvor et al. (2004). While modern AES implementations naturally benefit from heavily parallelized LUT-based S-boxes to achieve multi-gigabit speeds, legacy IDEA implementations relying on standard modular division typically struggle to match these contemporary rates. By replacing iterative division with the single-cycle Fermat reduction and zero-cost wire shifting, the proposed HLS-driven design achieves an exceptionally high throughput of 8.89 Gbps. This offers a massive improvement over classical hardware approaches and successfully bridges the performance gap with highly optimized modern ciphers.

Despite the highly efficient 3.41% LUT utilization, which suggests that dozens of encryption cores could theoretically be replicated on a single Artix-7 chip to multiply throughput, practical physical limitations apply. As the design is replicated in a multi-core configuration, routing the I/O interfaces to distributed memory banks introduces severe interconnect

congestion across the FPGA fabric. This routing density can drastically degrade the maximum achievable clock frequency (F_{max}). Therefore, physical board deployments must carefully integrate interconnect protocols (e.g., AXI-Stream) to prevent external memory bandwidth bottlenecks from negating the core's internal computational speed.

5. CONCLUSIONS AND FUTURE WORK

The transition to a hardware-centric paradigm for the IDEA cipher demonstrates the enduring relevance of symmetric cryptography when paired with spatial computing. Rather than merely accelerating a legacy algorithm, this work illustrates a fundamental architectural shift: overcoming the von Neumann bottleneck by localizing data processing allows resource-constrained platforms, such as the Artix-7 FPGA, to achieve multi-gigabit throughput. By sustaining an 8.89 Gbps data rate while consuming only 3.41% of the available logic fabric, this implementation establishes a highly efficient resource-to-throughput blueprint, achieving a Throughput-to-Area Efficiency (TAE) of 4.11 Mbps/LUT. This performance confirms the design's viability for deploying high-bandwidth secure channels in latency-sensitive edge computing and Internet of Things (IoT) environments, ensuring scalability and low-power operation.

The success of this architecture relies on resolving algorithmic bottlenecks directly at the physical synthesis level. The development of a Fermat-based division-free multiplier to eliminate iterative cycles, the use of static wire routing for key expansion, and the complete partitioning of memory arrays for simultaneous operand access collectively prove that complex mathematical transformations can be managed with optimal throughput (Initiation Interval $II = 1$), guaranteeing deterministic data processing despite the inherent depth of the encryption pipeline.

Furthermore, this research validates High-Level Synthesis (HLS) as a robust methodology for cryptographic hardware. Elevating the abstraction from traditional RTL to C++ significantly accelerated development through rapid design-space exploration without sacrificing bare-metal performance. Coupled with the successful verification of bidirectional encryption and decryption under real-world constraints, this work proves that aggressive spatial optimizations preserve cryptographic integrity, establishing a reliable framework for future hardware accelerators and modular system-on-chip (SoC) integration.

Building upon the methodologies established in this paper, several critical applications and architectural extensions are proposed for advanced future work on cryptographic accelerators:

- **Zero-Knowledge Proofs (ZKP)**: The arbitrary precision bit-slicing and spatial unrolling techniques demonstrated in this paper can be adapted to design custom FPGA accelerators for multi-scalar multiplications (MSM) and Number Theoretic Transforms (NTT) required for real-time ZKP generation, specifically by minimizing DSP exhaustion and eliminating dynamic routing penalties.
- **Homomorphic Encryption (HE)**: The memory partitioning strategies and division-free arithmetic optimizations developed for IDEA are directly transferable to HE. Future research will map HE algorithms onto FPGA fabrics to accelerate the massive polynomial multiplications that bottleneck general-purpose CPUs by ensuring single-cycle multi-port data access across distributed register files.
- **RISC-V and RTEU Integration**: A primary application involves extending the RISC-V instruction set (ISA) with dedicated security primitives based on the Fermat-based modular multiplier. Deploying these optimized cores within Real-Time Execution Units (RTEUs) will provide deterministic, low-latency environments essential for autonomous systems, enabling single-cycle cryptographic instruction execution without stalling the processor pipeline.

DECLARATION OF GENERATIVE AI AND AI-ASSISTED TECHNOLOGIES IN THE WRITING PROCESS

During the preparation of this work the authors used Google Gemini Pro in order to refine sentence phrasing and assist with LaTeX formatting and structural adjustments. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

REFERENCES

Beuchat, J.L. (2003). Modular multiplication for FPGA implementation of the IDEA block cipher. In *Proceedings of the 14th IEEE International Conference on Application-specific Systems, Architectures and Processors*, 412–422. IEEE. doi:10.1109/ASAP.2003.1231754.

Chodowiec, P. and Gaj, K. (2003). Very compact FPGA implementation of the AES algorithm. In *Cryptographic Hardware and Embedded Systems (CHES 2003)*, 319–333. Springer. doi:10.1007/978-3-540-45238-6_26.

Kastner, R., Matai, J., and Neuendorffer, S. (2018). *Parallel Programming for FPGAs*. arXiv. doi:10.48550/arXiv.1805.03648.

Lai, X., Massey, J.L., and Murphy, S. (1991). Markov ciphers and differential cryptanalysis. In *Advances in Cryptology — EUROCRYPT '91*, 17–38. Springer. doi:10.1007/3-540-46416-6_2.

Magyari, A. and Chen, Y. (2025). Hardware optimized modular reduction. *Electronics*, 14(3), 550. doi:10.3390/electronics14030550. Special Issue on Emerging Applications of FPGAs and Reconfigurable Computing Systems.

McIvor, C., McLoone, C., and McCanny, J.V. (2004). Modified montgomery modular multiplication and RSA exponentiation techniques. *IEE Proceedings - Computers and Digital Techniques*, 151(6), 402–408. doi:10.1049/ip-cdt:20040791.

National Institute of Standards and Technology (NIST) (2001). Advanced encryption standard (AES). Federal Information Processing Standards Publication (FIPS) 197, U.S. Department of Commerce. doi:10.6028/NIST.FIPS.197.

Peltier, A., Peneau, P.Y., and Novo, D. (2021). Evaluating High-Level Synthesis for Cryptography. In *Proceedings of the 2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. doi:10.1109/ICCAD51958.2021.9643444. Available at HAL: hal-03347174.

Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, 2nd edition.

Xilinx (2020). *7 Series FPGAs Data Sheet: Overview (DS180)*. Advanced Micro Devices (AMD).

Xilinx (2026). *Vitis High-Level Synthesis User Guide (UG1399)*. Advanced Micro Devices (AMD).

Zimmermann, R., Curiger, A., Bonnenberg, H., Kaul, H., Felber, N., and Fichtner, W. (1994). Vinci: A 177 Mb/s VLSI implementation of the International Data Encryption Algorithm. *IEEE Journal of Solid-State Circuits*, 29(3), 303–307. doi:10.1109/4.272135.